Chapter 4
Comparative Study on Mostly Used NoSQL Databases

1. Introduction

- Data Model, which defines how a database stores data and how it handles concurrent access.
- Query Model, in which the surveyed databases differ the most. This category contains the power of the used query language, its restrictions, and how it can be used
- Sharding and Replication.
- Consistency category, the consistency level the databases achieve and trade-offs they make are analysed.

2.1. SimpleDB

- Amazon SimpleDB is optimized to provide high availability and flexibility, with no administrative burden
- SimpleDB generates and handles multiple geographically distributed replicas of your data automatically to enable greater availability and data durability.
- It removes the demand on the Developers to consider data modeling, maintaining of index and optimizing performance (tuning), or data manipulation (also done automatically).

Data Model Hierarchal concept of SimpleDB.

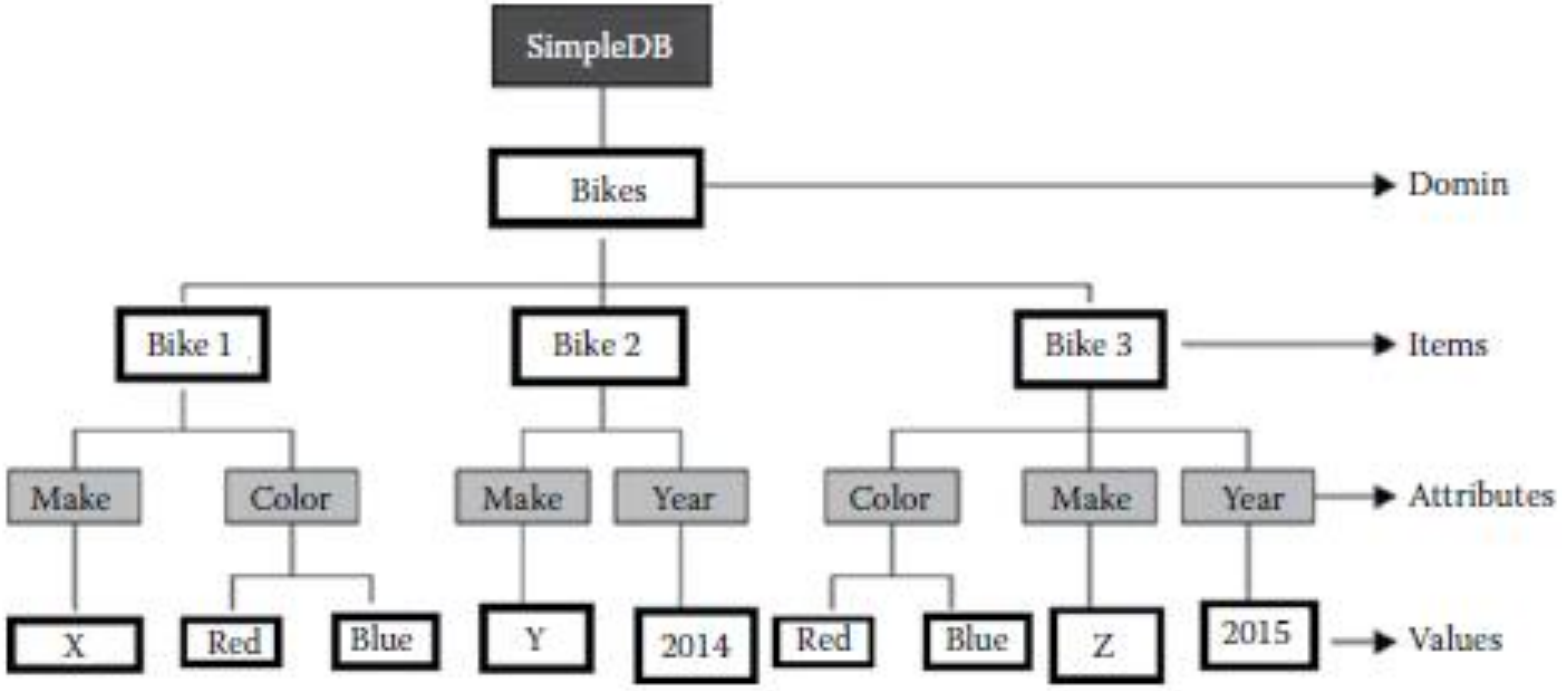


FIGURE 4.1
Hierarchal concept of SimpleDB.

Domain – container for structured data (worksheet in spreadsheet)
Items – like Rows on spreadsheet
NB Queries cannot run across domains
Items in domain completely separate from other domains

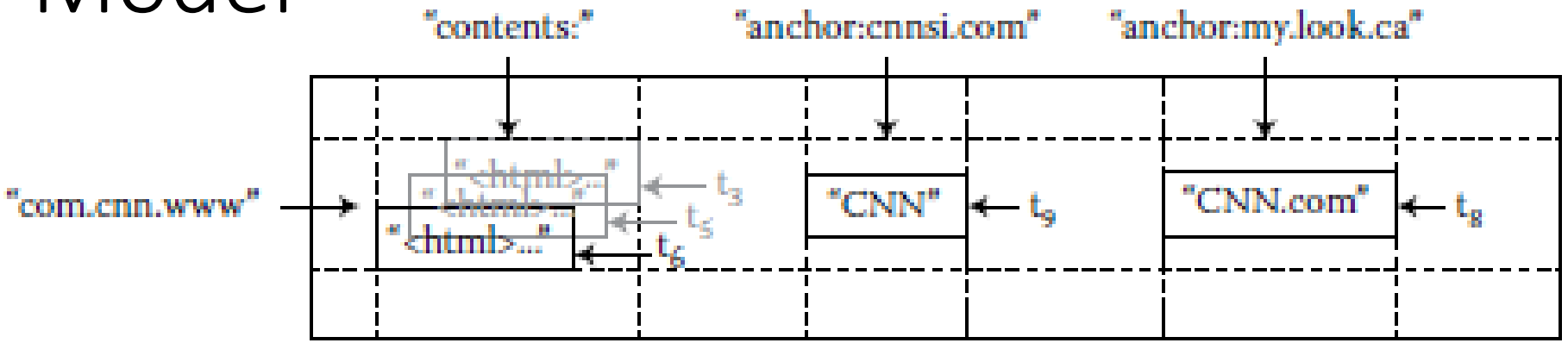
Consistency

- Multiple copies of each domain are stored in SimpleDB. A successful write guarantees that all copies of the domain will durably persist.
- An eventual read might not reflect the results of a recently completed write Consistency of read is usually reached within a short span of time.

2.2 Google’s BigTable

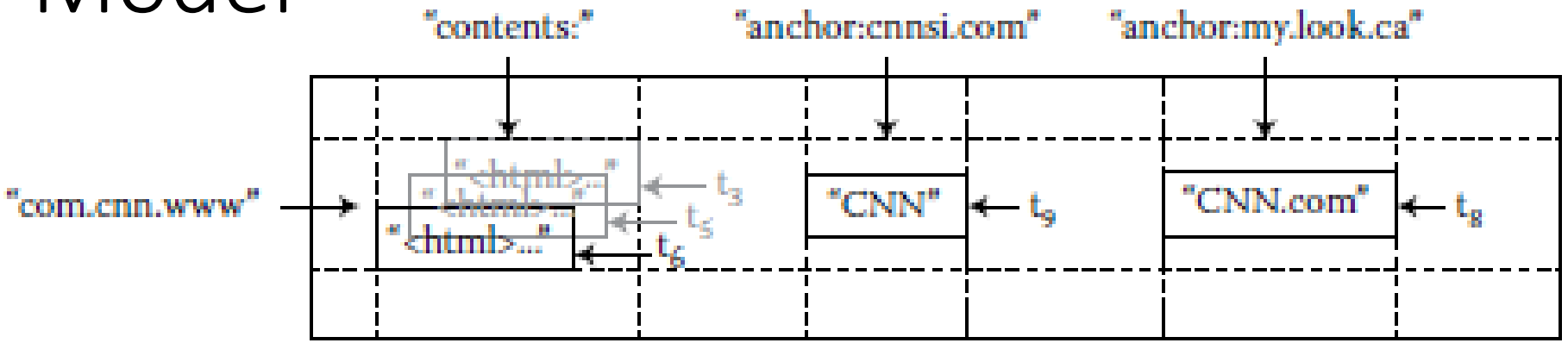
- BigTable is a light, scattered, constant, multidimensional, sorted map database.
- This map’s indexing is done by row key, column key, and a timestamp.
- In BigTable, value is determined as array of bytes. BigTable stores structured information. It does not impose any size restrictions for each value.
- BigTable is highly scalable and can range up to thousands of computers. BigTable is designed on top of Google File System. Chubby is used to store the root tablet, schema details, access control lists, coordinate, and identify tablet servers.
- The automatic clean-up process of BigTable is done by removing SST tables and unused data by using Mark and Sweep algorithm.

Data Model



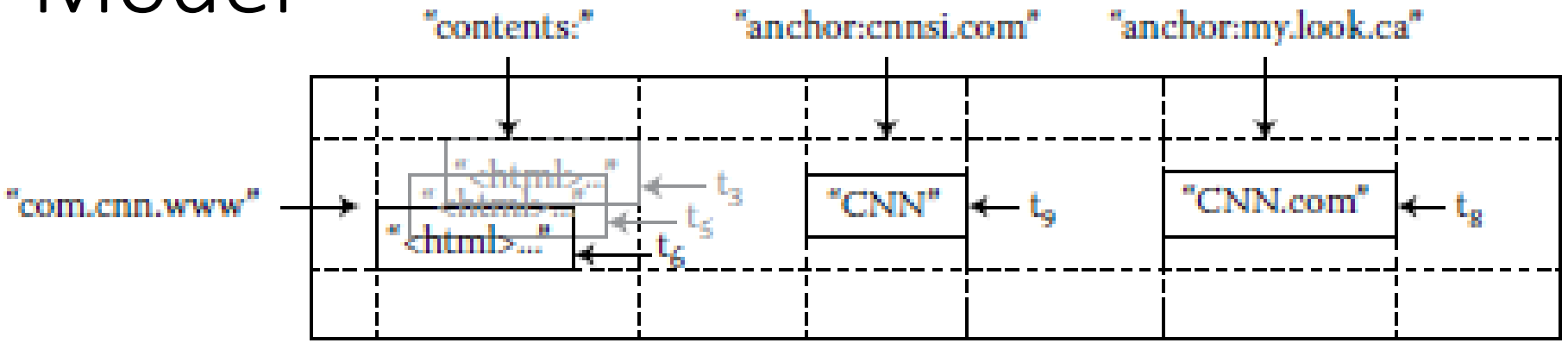
- The row keys in a table are arbitrary strings (currently up to 64 KB in size, although 10–100 bytes is a typical size for most of our users).
- Every read or write of data under a single row key is atomic, and a design decision makes it easier for clients to reason about the system's behaviour in the presence of concurrent updates to the same row.
- BigTable maintains data in lexicographic order by row key.
- The row range for a table is dynamically partitioned.
- Each row range is called a tablet, which is the unit of distribution and load balancing.
- As a result, reads of short row ranges are efficient and typically require communication with only a small number of machines.

Data Model



- Families of columns are used – in this example the name is anchor:
- Families Column keys are grouped into sets called column families, which form the basic unit of access control.
- All data stored in a column family are usually of the same type (we compress data in the same column family together).
- A column family must be created before data can be stored under any column key in that family; after a family has been created, any column key within the family can be used.
- It is our intent that the number of distinct column families in a table be small (in the hundreds at most) and that families rarely change during operation.
- In contrast, a table may have an unbounded number of columns. Column family names must be printable, but qualifiers may be arbitrary strings.

Data Model



- Each cell in a BigTable can contain multiple versions of the same data; these versions are indexed by timestamp.
- BigTable timestamps are 64-bit integers. They can be assigned by BigTable, in which case they represent “real time” in microseconds or be explicitly assigned by client applications.
- Applications that need to avoid collisions must generate unique timestamps themselves.
- Different versions of a cell are stored in decreasing timestamp order so that the most recent versions can be read first.
- To make the management of versioned data less onerous, we support two per-column-family settings that tell BigTable to garbage-collect cell versions automatically.
- The client can specify either that only the last ***n*** versions of a cell be kept or that only new-enough versions be kept.

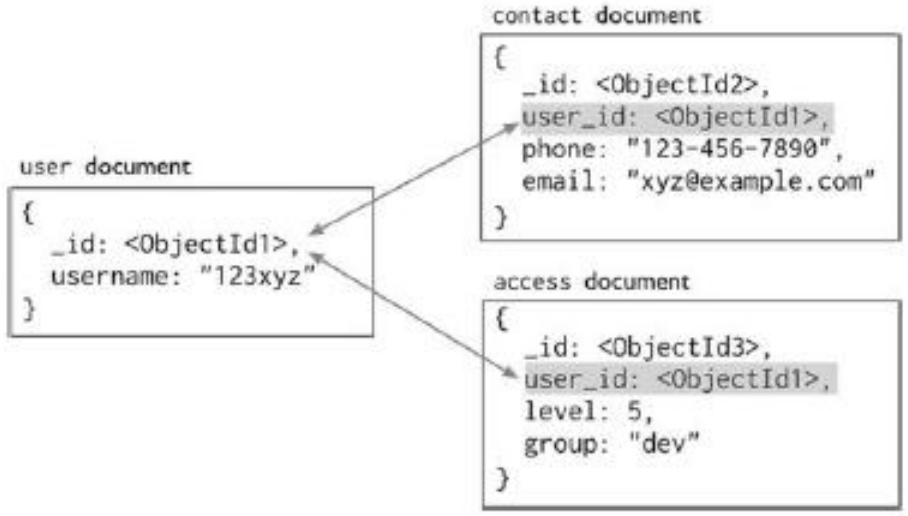
Sharding & Replication read section 4.2.2.3.1

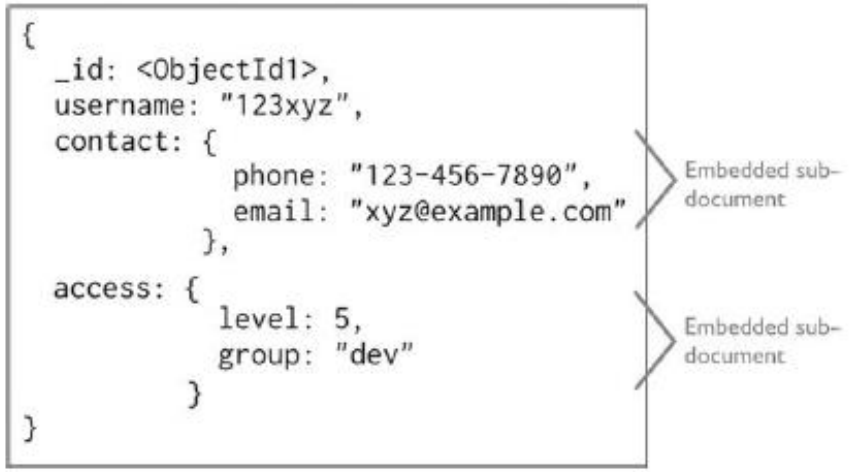
- Sharding: Shared nothing – partitioning scheme
- *temporary* failures – requires durability
- *permanent* failures – requires replication
- Most (but, interestingly, not all) distributed systems use *both* durability *and* replication to store data reliably.
- For instance, each data modification might be written to at least three disks. If one disk fails, the data are proactively copied onto a new disk so that at least three copies are usually available. That way, only three simultaneous permanent failures cause data loss.
- Sequential storage: Less mechanical movement in disks and predictable
- Organisation: Sparse hash table; uses strings as names of metadata – everything is a string
- Supports Put, get, scan and delete operations.

MongoDB

- Unlike other databases where you must determine schema before entering data, MongoDB has a flexible schema. The flexibility allows mapping of documents to an entity or an object. The key challenge n data modeling is balancing the needs of the application, the performance characteristics of the database engine, and the data retrieval patterns.
- Document Structure
- References store the relationships between data across documents by links or references.

MongoDB: Data Model





- Embedded documents capture relationships between data by storing related data in a single document structure.
- MongoDB documents make it possible to embed document structures in a field or array within a document.
- These *denormalized* data models allow applications to retrieve and manipulate related data in a **single database operation**

Data documents

- In MongoDB, write operations are atomic at the document level, and no single write operation can atomically affect more than one document or more than one collection.
- A denormalized data model with embedded data combines all related data for a represented entity in a single document. This facilitates atomic write operations since a single write operation can insert or update the data for an entity.

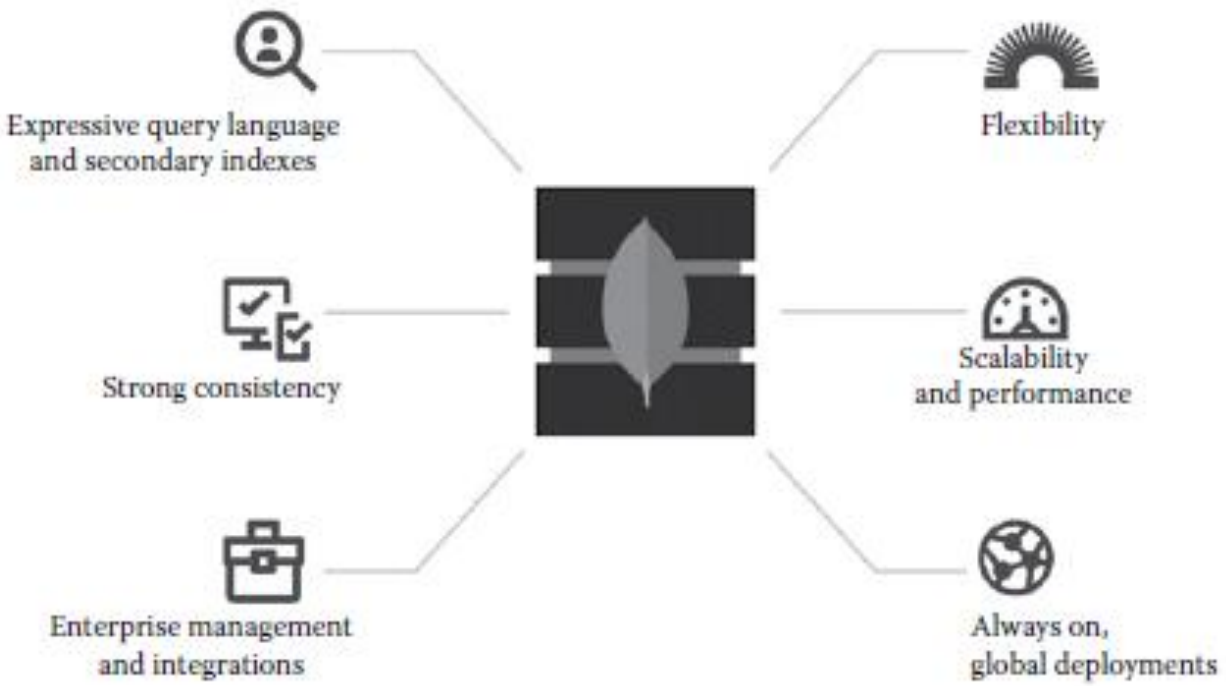
Sharding

- *Shards* store the data. To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set.
- *Query routers*, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. A client sends requests to a mongos, which then routes the operations to the shards and returns the results to the clients. A sharded cluster can contain more than one mongos to divide the client request load, and most sharded clusters have more than one mongos for this reason.
- *Configuration servers* store the cluster's metadata. These data contain a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards

Sharding and Partitioning

- *Data Partitioning:* MongoDB distributes data, or shards, at the collection level. Sharding partitions a collection's data by the *shard key*.
- *Shard Keys:* To shard a collection, you need to select a *shard key*. A shard key is either an indexed field or an indexed compound field that exists in every document in the collection. MongoDB divides the shard key values into *chunks* and distributes the chunks evenly across the shards. To divide the shard key values into chunks, MongoDB uses either *range-based partitioning* or *hash-based partitioning*.
- *Range-Based Sharding:*MongoDB divides the data set into ranges determined by the shard key values to provide *range-based partitioning*.
- *Hash-based partitioning*: MongoDB computes a hash of a field's value and then uses these hashes to create chunks.

Characteristics of MongoDB



- Know the meaning of each of these items: p 87-88 very important.
- Monotonic reads and writes
- No value older than the previous read will be returned
- Writes will be applied in sequece

Replication and Failure handling

- MongoDB handles replication through an implementation called “replication sets.” Replication sets in their basic form are somewhat similar to nodes in a master–slave configuration. A single primary member is used as the base for applying changes to secondary members.
- The difference between a replication set and master–slave replication is that a replication set has an intrinsic automatic failover mechanism in case the primary member becomes unavailable.

- *Primary member*: The primary member is the default access point for transactions with the replication set. It is the only member that can accept write operations.
- *Secondary members*: A replication set can contain multiple secondary members. Secondary members reproduce changes from the oplog on their own data.

- *Arbiter*: An arbiter is an optional member of a replication set that does not take part in the actual replication process. It is added to the replication set to participate in only a single, limited function: to act as a tiebreaker in elections.

In the event that the primary member becomes unavailable, an automated election process happens among the secondary nodes to choose a new primary. If the secondary member pool contains an even number of nodes, this could result in an inability to elect a new primary due to a voting impasse. The arbiter votes in these situations to ensure that a decision is reached.

CouchDb

1. *Data model*: CouchDB implements a Document data model.

2. No *Fixed schema*: Many NOSQL databases do not enforce a fixed schema definition for the data store in the database. This enables changes in data structures to be smoothly evolved at the database level over time, enhancing modifiability. Typically, objects with different formats can be stored together in the same collections. In CouchDB, a fixed schema definition **is not required**.

3. No *Opaque data objects required*: A database may store data as opaque objects that require interpretation in the client that issues a query. Opaque data objects usually require embedded metadata for interpretation stored with every object, and this may lead to “size bloat” in a database, negatively impacting performance.
4. *Hierarchical data objects*: A database support many hierarchical data structures, often known as sub-objects or documents. Graph databases support hierarchical databases naturally in their data model. In CouchDB, sub-objects are supported.

CouchDb

Apache CouchDB is an open-source document-oriented NoSQL database, implemented in the concurrency-oriented language Erlang; it uses JSON to store data, JavaScript as its query language using MapReduce, and HTTP for an API

Main features [edit]

ACID Semantics

CouchDB provides ACID semantics.[10] It does this by implementing a form of Multi-Version Concurrency Control, meaning that CouchDB can handle a high volume of concurrent readers and writers without conflict.

Built for Offline

CouchDB can replicate to devices (like smartphones) that can go offline and handle data sync for you when the device is back online.

Distributed Architecture with Replication

CouchDB was designed with bi-directional replication (or synchronization) and off-line operation in mind. That means multiple replicas can have their own copies of the same data, modify it, and then sync those changes at a later time.

Document Storage

CouchDB stores data as "documents", as one or more field/value pairs expressed as JSON. Field values can be simple things like strings, numbers, or dates; but ordered lists and associative arrays can also be used. Every document in a CouchDB database has a unique id and there is no required document schema.

Eventual Consistency

CouchDB guarantees eventual consistency to be able to provide both availability and partition tolerance.

Map/Reduce Views and Indexes

The stored data is structured using views. In CouchDB, each view is constructed by a JavaScript function that acts as the Map half of a map/reduce operation. The function takes a document and transforms it into a single value that it returns. CouchDB can index views and keep those indexes updated as documents are added, removed, or updated.

HTTP API

All items have a unique URI that gets exposed via HTTP. It uses the HTTP methods POST, GET, PUT and DELETE for the four basic CRUD (Create, Read, Update, Delete) operations on all resources.

CouchDB also offers a built-in administration interface accessible via Web called Futon.[11]